jet INTL JOURNAL OF ELECTRONICS AND TELECOMMUNICATIONS, 2013, VOL. 59, NO. 4, PP. 357–362
Manuscript received November 12, 2013; revised December, 2013. DOI: 10.2478/eletel-2013-0043

Experimental Comparison of Synthesis Tools Altera Quartus II and Synthagate

Marek Węgrzyn and Andrei Karatkevich

***Abstract*—The paper presents comparison between efficiency of an industrial FPGA design software tool Altera Quartus II and similar design software tool Synthagate by Syntezza company of an academic origin. The experiments were performed using a series of examples describing the Mealy finite state machines; one-hot state encoding was used in all cases. Area (number of used logical blocks) was the main parameter used for the comparison. Influence of the way of FSM description (in VHDL language) on the quality of synthesis was studied. The obtained results show that Synthagate in almost all cases performs synthesis more efficiently and essentially quicker than Altera Quartus. Section I presents motivation of the research. Section II reminds the notion of FSM. Section III describes problems which had to be solved to provide correctness of experimental comparison. Section IV presents some details about state encoding way used in the experiments. In Section V, the experimental results are presented. Section VI describes the problems related to the comparison which still have to be solved. Section VII presents the conclusions from the experiments. Section VIII suggests possible reasons of the detected situation.**

***Keywords*—Logic design, state machines, logic devices, FPGA, VHDL**

I. Introduction

DESIGNERS of the CAD systems of an academic origin often claim that their tools are much more efficient than the industrial CAD tools both in size and speed of the obtained designs [1]–[5]. Some of the experiments described in the publications show that area of the devices constructed by the academic and industrial tools differs several times in average. It is demonstrated among others in [3], where an academic tool DEMAIN, developed in the Warsaw University of Technology, is compared with the industrial tools, such as Altera MAX+Plus II, Quartus II and Leonardo Spectrum. Gain in size provided by DEMAIN in comparison with the industrial tools is about two times in average for sequential devices and several times for combinational logic.

The authors decided to perform some experiments for checking such claims. Two systems have been experimentally compared: a popular industrial tool Altera Quartus II (version 12) [6] and tool Synthagate by Synthezza company [7], which uses the algorithms developed in Bar Ilan University (Israel) by the team lead by S. Baranov [8]. The sequential FPGA devices were synthesized, specified by the finite state machine descriptions.

M. Wegrzyn and A. Karatkevich are with the Institute of Computer Engineering and Electronics, University of Zielona Góra, Z. Szafrana 2, 65-516 Zielona Góra, Poland (e-mails: {M.Wegrzyn; A.Karatkevich}@iie.uz.zgora.pl).

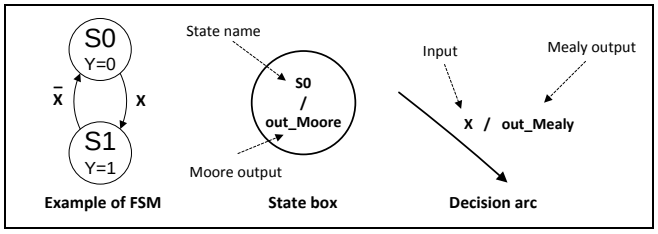

Fig. 1. Elements of finite state machines.

II. Finite State Machines

Finite State Machine (FSM) [9] is a mathematical model that describes a digital system with memory, i.e. a sequential logic circuit. It is an abstract machine that at each moment of (discrete) time can be in one of a finite numbers of states, obtain one of a finite numbers of possible inputs and generate one of a finite number of outputs. Next state is a function of current state and current input; current output is a function of current state (Moore automaton [10]) or a current state and current input (Mealy automaton [11]). Elements of the mentioned models are shown in Fig. 1; for the details see [9], [12], [13]. In our experiments we have used the Mealy state machines, which are (in a sense) more general than the Moore automata.

A finite state machine can be represented, among others, as a state transition table (Fig. 2) or as a state diagram (Fig. 3).

III. Notes on the Way of Specification

The first experiments demonstrated that Synthagate seems to be much more efficient synthesis tool than Quartus, but it could be caused partially by the fact that two compared tools interpret the VHDL descriptions in different ways.

Synthagate obtains its input description of an FSM in a KISS2-like text format, as shown in Fig. 2 (the corresponding graph representation of the FSM is shown in Fig. 3).

There are many ways to describe FSMs as synthesizable and efficient HDL models. A traditional representation of Mealy and Moore state machines as circuits consists of three main blocks: next state block, outputs block and register (memory block). In Mealy machine its outputs are determined by the current state and the inputs. In Moore machine its outputs are determined only by its current state. For VHDL, *process* statement is the most suitable way for describing FSMs. Designers may specify several processes, depending on how the different parts of the model are considered and decomposed. General information about all models of automata is systematized in Fig. 4. However, each description of automaton must include:

```
a1   a2   x6          y8y9      1
a1   a5   ~x6*x7      y6        2
a1   a5   ~x6*~x7     y3y6y10   3
a2   a2   x4*x1       y1y2      4
a2   a6   x4*~x1      y3y4      5
a2   a4   ~x4         y4        6
a3   a6   1           y3y4      7
a4   a1   x5          --        8
a4   a2   ~x5*x1      y8y9      9
a4   a6   ~x5*~x1     y3y4      10
a5   a2   x1*x2*x3    y1y3      11
a5   a3   x1*x2*~x3   y6y7      12
a5   a2   x1*~x2      y1y3      13
a5   a4   ~x1         y4        14
a6   a5   x6          y6y7      15
a6   a6   ~x6         y3y4      16
```

Fig. 2. An example of FSM description in Synthagate's input format.

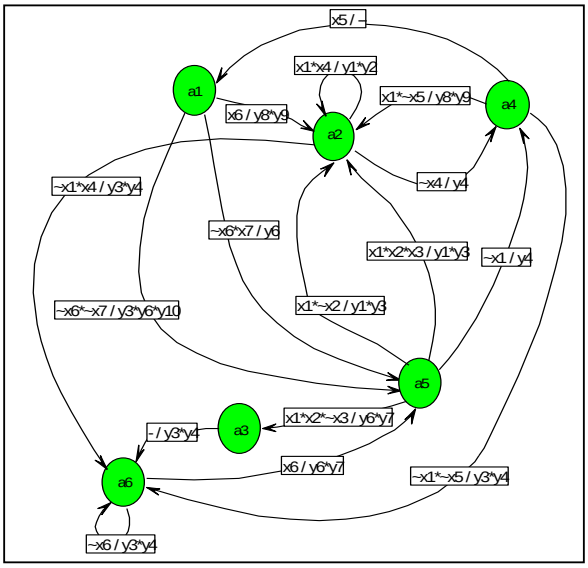

Fig. 3. Graph representation of the FSM corresponding to the description from Fig. 2.

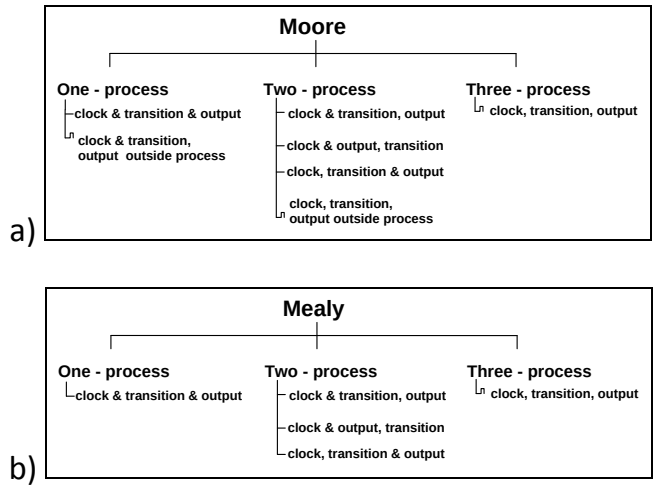

Fig. 4. Structure of state machine HDL models: a) Moore type, b) Mealy type.

```
begin
   process (clk, rst)
   procedure proc_Girl10 is
   begin
     y1   <= '0';  y2   <= '0';
     y3   <= '0';  y4   <= '0';
     y6   <= '0';  y7   <= '0';
     y8   <= '0';  y9   <= '0';
     y10  <= '0';

   case current_Girl10 is
   when s1 =>
      if ( x6 ) = '1' then
         y8 <= '1';
         y9 <= '1';
         current_Girl10 <= s2;
      elsif (not x6 and x7) = '1' then
         y6 <= '1' ;
         current_Girl10 <= s5;
      else
         y3 <= '1';
         y6 <= '1';
         y10 <= '1';
         current_Girl10 <= s5;
      end if;
   ...
   end process;
```

Fig. 5. A fragment of an FSM description in VHDL (Synthagate).

- variable, which value is defined by a state of automaton;
- clock;
- specification of the state transition flow;
- specification of outputs;
- reset (synchronous or asynchronous).

Synthagate converts an FSM state transition table (Fig. 2) into a VHDL or Verilog description, using the strictly reduced subsets of these languages, which contain only the synthesizable constructs. Fragment of such description (in VHDL) is shown in Fig. 5. The problem is that Altera Quartus interprets this description in such a way that there is a flip-flop added for every output signal in a synthesized device. That happens because the values of the signals are assigned inside a process having a clock signal (CLK) on its sensitivity list [12].

However, the Synthagate tool synthesizes for such descriptions the devices with purely combinational outputs. Naturally, such devices have less flip-flops and, in many cases, less logical blocks. That means that different VHDL descriptions should be used for two tools to ensure purity of the experiments.

Values can be assigned to the output signals outside the processes having CLK in the sensitivity list (in another process or outside the processes). For this reason specifications used for Quartus were changed in such a way that the values are assigned to the output signals outside the process which has the clock signal on its sensitivity list. A fragment of such description (of the same automaton as described in Fig. 5) is shown in Fig. 6.

The experiments with Quartus II show that such description leads to synthesis of the devices with combinational outputs and of less area. However, another problem was detected related to such variant of specification. The devices synthesized by Altera Quartus using such description have separate

```
FSM_machine_1: process (clk, reset)
  begin
    if reset='1' then
       stan <= st1;
    elsif clk'event and clk = '1' then
     case stan is
       when st1 =>
          if x(6) = '1' then
               stan <= st2;
          elsif x(6) = '0' and x(7) = '1' then
               stan <= st5;
          elsif x(6) = '0' and x(7) = '0' then
               stan <= st5;
          end if;
       when st2 =>
   ...
  end process;

Y_assignment:
  y <=    "0000000110"
          when (stan=st1 and (x(6)='1'))
     else "0000010000"
          when (stan=st1 and (x(6)='0' and x(7)='1'))
     else "0010010001"
          when (stan=st1 and (x(6)='0' and x(7)='0'))
     else
     ...
```

Fig. 6. A fragment of an alternative FSM description (variant 1).

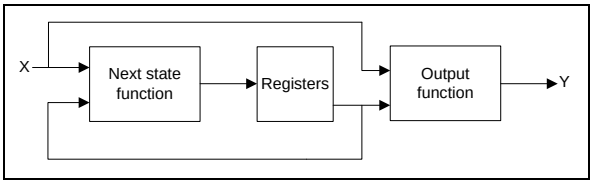

Fig. 7. Structure of a device designed by Altrera Quartus from the specification shown in Fig. 6.

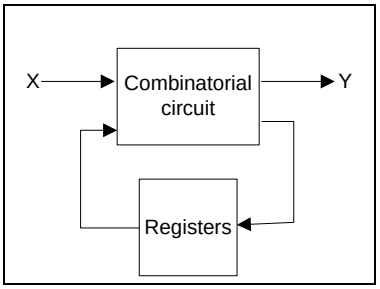

Fig. 8. Structure of a device designed by Synthagate.

combinational parts generating output signals and input signals for the flip-flops. Such device has a structure as shown in Fig. 7 At the same time Synthagate performs common minimization of both output and next state functions, which allows reducing the device area. This leads to the device structure shown in Fig. 8. Such structure requires less logical blocks (note that it is true only for Mealy FSMs) [13], [14].

Common minimization of two systems of Boolean functions requires (according to the interpretation by Altera Quartus) description of all of them in the same process. However, as far as the outputs of the device have to be purely combinational (without the flip-flops), such process should not have a clock signal in its sensitivity list. That can be obtained by means of introducing an additional signal, which is intended for

```
Outputs_States: process (stan, x)
begin
    case stan is
       when st1 =>
            if x(6) = '1' then
               stan_temp <= st2; y <= "0000000110";
            elsif x(6) = '0' and x(7) = '1' then
               stan_temp <= st5; y <= "0000010000";
            elsif x(6) = '0' and x(7) = '0' then
               stan_temp <= st5; y <= "0010010001";
            else stan_temp <= st1; y <= "0000000000";
            end if;
       ...
end process;

FSM_machine_2: process (clk, reset)
begin
    if reset='1' then
        stan <= st1;
     elsif clk'event and clk = '1' then
        stan <= stan_temp;
     end if;
end process;
```

Fig. 9. A fragment of an alternative FSM description (variant 2).

remembering a next FSM state, and an additional process, which assigns the value of the mentioned signal to the variable describing a current state of the automaton. This new process has to have CLK in its sensitivity list.

For this reason one more variant of the specification was developed, shown in Fig. 9. Such description allows to obtain much better synthesis results, using Altera Quartus II. For the considered example (Fig. 3) result of synthesis by Quartus consists of 34 logical blocks, if the description shown in Fig. 6 is used, and only 23 logical blocks, if the same automaton is described as shown in Fig. 9 (when "area" was selected as the main optimization criterion).

This variant was used for the comparison, as far as it seems that such approach allows to compare two tools in the proper way.

IV. Notes on the State Encoding

There are several methods of state encoding of the finite state machines. Generally they can be divided into the methods trying to minimize the number of the used flip-flops (the minimal necessary number of the flip-flops is $\lceil log_2 n \rceil$, where n is the number of states) and the one-hot encoding (where number of required flip-flops is the same as the number of the states. Normally at each state one flip-flop is set, and all other flip-flops are reset). The classical synthesis methods usually used the binary encoding and minimized number of flip-flops; however, for the FPGA synthesis it is common to use one-hot state encoding [15], because in such case the excitation functions are simpler, and for the FPGA structures it means lower number of logical blocks in most cases. Most synthesis tools allow user to select a state encoding method, and in the FPGA synthesis tools such as Altera Quartus II the one-hot encoding is usually the default encoding method. For the details on influence of state encoding on speed and hardware consumption see, for example, [14], [16], [17].

Methods of representing of states in VHDL language are described in detail in [13], [16], [18]. For the one-hot state encoding it is especially convenient to use the enumeration

```
type name is ( state [, state] );

type StateType is (S0, S1, S2, S3);
…
signal STATE: StateType;
…
STATE <= S2;
```

Fig. 10. Example of applying enumeration type to description of states.

TABLE I
PARAMETERS OF THE TEST SAMPLES

Group	Number of inputs	Number of outputs	Number of states	Number of transitions
Small	5 – 43	8 – 30	11 – 47	37 – 143
Medium	16 – 65	13 – 209	17 – 274	111 – 712
Large	45 – 65	24 – 209	79 – 274	383 – 1564
Great	63 – 68	54 – 112	174 – 1275	4900 – 10980
SuperGreat	70 – 75	70 – 112	199 – 1806	24422 – 84993

data type (Fig. 10). Such approach was used in the described experiments.

V. EXPERIMENTAL RESULTS

For the experiments a series of state machine descriptions with binary inputs and outputs was used. Some of the examples were otained from the academic sources (partially from the student projects), some others are practical (industrial). The examples were divided into 5 groups differing in their size: *Small*, *Medium*, *Large*, *Great*, and *SuperGreat*, with average number of states about 30, 100, 200, 700 and 1000, respectively. Main qualitative parameters of the examples belonging to different groups are presented in Tab. I. For all examples and both tools the one-hot state encoding method was used and area as the main optimization criterion was selected. The input specifications for Quartus were generated using variant 2 presented above (Fig. 9).

Results of experiments for several examples from different groups are presented in Tab. II and Tab. III. They also demonstrate comparison between numbers of FPGA logic blocks in the devices designed by two tools. Table II presents comparison of the Synthagate results and the Quartus results for the case, when the first (shown in Fig. 6; the converter of the formats was developed as a part of research work by Bukowiec [19]) variant of VHDL specification was used for Quartus; Table III, correspondingly, presents such comparison for the second variant (Fig. 9).

Table IV presents average results for all groups of examples (95 FSM descriptions were used for the experiments), with the second variant of VHDL specification. It is easy to see that ratio of the size of the devices designed by Quartus and Synthagate decreases with growing size of the examples. For the group of biggest automaton which Quartus was able to proceed, designs by Synthagate use in average 4 times less logical blocks than designs by Quartus.

The obtained results demonstrate definitely, that Synthagate is a more efficient and deeply optimizing tool for designing sequential FPGA devices than Altera Quartus II. It is also worth noting that Synthagate performs synthesis essentially

TABLE II
SELECTED RESULT OF THE EXPERIMENTS; VHDL DESCRIPTION AS IN FIG. 6

Example	Group	Number of blocks (Quartus)	Number of blocks (Synthagate)	Ratio of area (%)
girl10	Small	34	19	56%
lift2	Small	127	48	38%
cpu	Small	165	38	23%
sara	Medium	470	114	24%
bs	Medium	729	170	23%
gol	Medium	956	141	15%
yaron	Medium	2151	231	11%
mars2M	Medium	2243	422	19%
bulln	Large	5329	946	18%
exxm	Large	6456	990	15%
mars2	Large	3500	566	16%
bigm2r	Great	–	3948	–
zoom	Great	–	4481	–
group15m	Great	–	5933	–
otherm	Great	–	6772	–

TABLE III
SELECTED RESULT OF THE EXPERIMENTS; VHDL DESCRIPTION AS IN FIG. 9

Example	Group	Number of blocks (Quartus)	Number of blocks (Synthagate)	Ratio of area (%)
girl10	Small	23	19	83%
lift2	Small	61	48	79%
cpu	Small	80	38	48%
sara	Medium	233	114	49%
bs	Medium	368	170	46%
gol	Medium	449	141	31%
yaron	Medium	848	231	27%
mars2M	Medium	803	422	53%
bulln	Large	2305	946	41%
exxm	Large	3151	990	31%
mars2	Large	1312	566	43%
bigm2r	Great	12622	3948	31%
zoom	Great	17326	4481	26%
group15m	Great	28468	5933	21%
otherm	Great	26481	6772	26%

TABLE IV
AVERAGE RESULTS OF THE EXPERIMENTS FOR THE GROUPS OF DIFFERENT SIZE

Group	Average number of blocks (Quartus)	Average number of blocks (Synthagate)	Area (%)
Small	55	35	64%
Medium	540	216	40%
Large	2253	834	37%
Great	21224	5284	25%
SuperGreat	–	18313	–

quicker than Quartus. Quartus was unable to perform synthesis for the examples of the group *SuperGreat*, when the second variant of VHDL description was used; when the first variant was used, Quartus was unable to perform synthesis for both groups of biggest automaton – *Great* and *SuperGreat*. Synthagate needed only few seconds for each of the examples belonging to the group *Great* and few minutes for each of *SuperGreat* examples.

VI. Further Research

The experiments performed up to now do not provide the answers for all the questions arising in the considered research area. The following topics should be investigated for better understanding of differences between two kinds of the design tools.

1) The performed experiments were intended to compare area of the synthesized devices, but comparison of speed was not performed yet. Such comparison would be especially interesting for the case when speed is selected as the main optimization criterion in both tools. Synthagate allows calculating for the synthesized devices the critical path length, which limits maximal possible frequency of clock signal and hence maximal speed of the device. Calculating of critical path for the Quartus synthesis results would allow comparison of speed. According to the results obtained up to now, Synthagate with selected option of speed optimization generates in most cases the devises consisting of less number of logic blocks, then Altera Quartus with the option of area optimization.
2) To be sure that the results of Synthagate synthesis can be implemented in real hardware, it should be checked whether the blocks in the designs can indeed be connected by wires and switches in an FPGA structure. Certain doubts are caused here by the fact that Synthagate seems to consider combinational elements and flip-flops of the designs independently, and in the FPGA structures they are integrated into the logic blocks. Synthagate allows generating the netlists in EDIF format, which can be used for further steps of synthesis, performed by other design tools, but actually it has only the libraries for selected Xilinx FPGA families, not yet for Altera FPGAs. That means that checking possibility of the implementation of the results of Synthagate synthesis in the Altera FPGAs requires additional research.
3) Having the presented results of the experiments, an important question arises: what differences in the structures of the synthesized devices cause such a remarkable difference in their sizes? To answer this question, detailed analysis of the structure of the designs should be performed. The authors hope that such analysis can be made at least for the relatively simple examples, and that it would allow to understand which steps of the optimization are not performed successfully by the commercial design tools.
4) Comparison between synthesis time for both tools seems to be interesting and important. The results obtained up to now demonstrate that synthesis times may have the orders of magnitude difference, especially for the biggest test examples. The authors suppose that detailed analysis will allow understanding where is the bottleneck of the industrial tool, however the difficulty is that we usually do not know how long the separate steps of synthesis process are executed. Maybe analysis of the middle-size examples for which synthesis time differs dramatically for both tools will help to study the causes of the situation.
5) In the performed research VHDL language was used. Both considered tools allow to use Verilog instead. It would be useful to study the difference between two tools when Verilog is selected as a hardware design language. Whether different styles of Verilog description affect essentially the design quality, as it happens in the case of VHDL descriptions? Studying this question is planned as one of the directions of future research.
6) The performed experiments are limited to two synthesis tools. It would be interesting and useful to make similar experiments with different CAD tools, such as for example Xilinx ISE.

VII. Concluding Remarks

The research demonstrates that the CAD tool of an academic origin solves the tasks of synthesis of sequential FPGA devices more efficiently and quicker than one of the most popular industrial synthesis tools. The experiments also show that the difference is especially remarkable for the bigger examples. An intriguing question is: why big corporations produce relatively inefficient design tools, when it is possible to use the algorithms that are quicker and provide better optimization? Obviously, the industrial systems have richer and more user-friendly interfaces, huge libraries and many other features, which lack the tools created by small companies or university teams. On the other hand, tools like Synthagate allow their user to have more information about structure of the designed devices. Both kinds of tools have their good and bad sides. But it still does not explain why the corporations, in spite of competition between them, do not use the efficient algorithms of synthesis.

VIII. An Attempt of Explanation

The fact that commercial computer aided design tools are often less efficient than they could be if they would use different (known) algorithms of synthesis and optimization is known widely enough. Certainly a situation is possible such that an algorithm of an academic origin does not take into account some practically important details and, being theoretically efficient, cannot be applied for real-life projects. However, certain experiments (such as presented in this paper or in [3], [4]) demonstrate that the complete processes of FPGA synthesis can be performed much better (and quicker) than the commercial tools do that.

One of the possible explanations is that the most known electronic design automation software, such as Altera Quartus or Xilinx ISE, is produced by the corporations which are first of all the manufacturers of the programmable logic devices

(such as Altera Corporation or Xilinx). Their income depends mostly on the PLDs they sell, not on the computer-aided design tools. They are able to make their design automation software cheap and in some cases even free. Altera Quartus II, for example, has a free version (Quartus II Web Edition) which provides compilation and programming for a limited, however wide number of Altera devices. Quartus II Subscription Edition is also available for free download, with restricted functionality (for full functionality a user has to pay for a license). A very similar approach (free tools with limited functionality and a paid license for full functionality) uses Xilinx with its Xilinx ISE software.

The small companies of the academic origin cannot allow themselves to provide free or very cheap software; on the other hand, their tools usually do not have such rich interface and such wide support of different families of the programmable logic devices as the commercial tools provide. That is the reason why the tools of academic origin have a very weak position in competition with the commercial tools. In fact, the market of the electronic design automation software is dominated by a small number of corporations (such situation can be described as an oligopoly; however, unlike in a typical oligopoly, it leads not to higher costs of the sold goods (software tools), but to lower quality of them).

The electronic companies earning by selling the FPGAs and other integrated circuits are not interested in deep minimization of the electronic devices, because such minimization would reduce their benefits. Controlling the market of the design tools, they do not allow such optimization to be widely applied. That is one of the "conspiracy theories" explaining why the most efficient optimization algorithms are ignored by the big manufacturers of reconfigurable digital circuits and are not used in the software tools which dominate the market. However, it does not provide an idea of how this situation, which is certainly not good for the consumers, can be changed.

REFERENCES

[1] S. Baranov, "ASMs in high level synthesis of EDA tool Synthagate," in *Proceedings of the 4th IFAC Workshop on Discrete-Event System Design*, 2009, pp. 195–200.

[2] H. Belhadj, L. Gerbaux, M.-C. Bertrand, and G. Saucier, "Specification and synthesis of communicating finite state machines," in *Synthesis for Control Dominated Circuits*. Elsevier Science Publishers B.V., North-Holland, IFIP, 1993, pp. 91–102.

[3] M. Rawski, T. Luba, Z. Jachna, and P. Tomaszewicz, "The influence of functional decomposition on modern digital design process." in *Design of Embedded Control Systems*. Springer-Verlag, 2005, pp. 193–204.

[4] M. Rawski, P. Tomaszewicz, G. Borowik, and T. Łuba, "Logic synthesis method of digital circuits design for implementation with Embedded Memory Blocks of FPGAs," in *Design of Digital Systems and Devices*, ser. Lecture Notes in Electrical Engineering. Berlin Heidelberg: Springer-Verlag, 2011, vol. 79, pp. 121–144.

[5] A. Barkalov, L. Titarenko, and O. Hebda, "Optimization of Moore finite-state-machine matrix circuit," *Pomiary, Automatyka, Kontrola*, vol. 57, no. 8, 2011.

[6] *Design Software*, Altera. [Online]. Available: http://www.altera.com/products/software/sfw-index.jsp

[7] Synthezza, "Synthagate Overview." [Online]. Available: http://synthezza.com/synthagate-overview

[8] S. Baranov, *Logic and System Design of Digital Systems*. Tallinn: TUT Press, 2008.

[9] J. Hopcroft and J. Ullman, *Introduction to Automata Theory, Languages, and Computation*. Addison-Wesley, 1979.

[10] E. Moore, "Gedanken-experiments on sequential machines," *Automata Studies, Annals of Mathematical Studies*, vol. 34, pp. 129–153, 1956.

[11] G. Mealy, "A method to synthesizing sequential circuits," *Bell System Technical Journal*, pp. 1045–1079, 1955.

[12] M. Zwolinski, *Digital System Design with VHDL*. Pearson Education Limited, 2004.

[13] S. Chmielewski and M. Węgrzyn, "Modelling and synthesis of automata in HDLs," *Proceedings of SPIE : Photonics Applications in Astronomy, Communications, Industry and High-Energy Physics Experiments 2006*, vol. 6347, p. 14, 2006.

[14] S. Golson, "State machine design techniques for Verilog and VHDL," *Synopsys Journal of High-Level Design*, 1994. [Online]. Available: http://www.trilobyte.com/pdf/golson_snug94.pdf

[15] *HDL Synthesis for FPGAs Design Guide*, Xilinx, 1995.

[16] Lattice Semiconductor, "HDL Synthesis Coding Guidelines for Lattice Semiconductor FPGAs," 2005. [Online]. Available: http://www.latticesemi.com/lit/docs/technotes/tn1008.pdf

[17] T. Luba, *Computer design of digital circuits*. Warsaw: WKL, 2000, (in Polish).

[18] K. Skahill, *VHDL for programmable logic*. Addison-Wesley Publishing, 1996.

[19] A. Bukowiec, "Synthesis of finite state machines for FPGA devices based on architectural decomposition," Ph.D. dissertation, Uniwersytet Zielonogórski, 2009.